Security Scheduling for Real-world Networks

Manish Jain[*], Vincent Conitzer[†], Milind Tambe[*]
[*] Computer Science Department, University of Southern California, Los Angeles, CA. 90089
{manish.jain,tambe}@usc.edu
[†] Department of Computer Science, Duke University, Durham, NC. 27708
{conitzer}@cs.duke.edu

ABSTRACT

Network based security games, where a defender strategically places security measures on the edges of a graph to protect against an adversary, who chooses a path through a graph is an important research problem with potential for real-world impact. For example, police forces face the problem of placing checkpoints on roads to inspect vehicular traffic in their day-to-day operations, a security measure the Mumbai police have performed since the terrorist attacks in 2008. Algorithms for solving such network-based security problems have been proposed in the literature, but none of them scale up to solving problems of the size of real-world networks.

In this paper, we present SNARES, a novel algorithm that computes optimal solutions for both the defender and the attacker in such network security problems. Based on a double-oracle framework, SNARES makes novel use of two approaches: *warm starts* and *greedy responses*. It makes the following contributions: (1) It defines and uses `mincut-fanout`, a novel method for efficient warm-starting of the computation; (2) It exploits the submodularity property of the defender optimization in a greedy heuristic, which is used to generate "better-responses"; SNARES also uses a better-response computation for the attacker. Furthermore, we evaluate the performance of SNARES in real-world networks illustrating a significant advance: whereas state-of-the-art algorithms could handle just the southern tip of Mumbai, SNARES can compute optimal strategy for the entire urban road network of Mumbai.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

Algorithms, Security, Performance

Keywords

Game theory, Double oracle, Zero-sum games, Minimax

1. INTRODUCTION

Game theory provides the sound mathematical tools for reasoning about both defenders and attackers in security domains. Many algorithms, e.g. RANGER [12], GRANDE [13] have been developed on the principles of game theory to provide optimal scheduling strategies to the defenders to protect a set of diverse targets from adaptive intelligent adversaries. These algorithms have also seen many real-world deployments, including recently in the PROTECT system for the US Coast Guard [11].

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, Ito, Jonker, Gini, and Shehory (eds.), May 6–10, 2013, Saint Paul, Minnesota, USA.
Copyright © 2013, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

This paper models the urban network security problem as a problem with two players: a defender and an attacker. The pure strategies of the defender correspond to allocations of resources to edges in the network – for example, an allocation of police checkpoints to roads in the city. The pure strategies of the attacker correspond to paths from any *source* node to any *target* node – for example, a path from a landing spot on the Mumbai coast to the Mumbai airport.

In this domain, the strategy space of the defender grows exponentially with the number of available resources, whereas the strategy space of the attacker grows exponentially with the size of the network. For example, in a fully connected graph with 20 nodes and 190 edges, the number of defender actions for only 5 resources is $\binom{190}{5} \approx 2$ billion, while the number of possible attacker paths without any cycles is $O(10^{18})$. Real-world networks are significantly larger, e.g., the entire road network of the city of Mumbai has $9,503$ nodes (intersections) and $20,416$ edges (streets), and the security forces can deploy large number of resources.

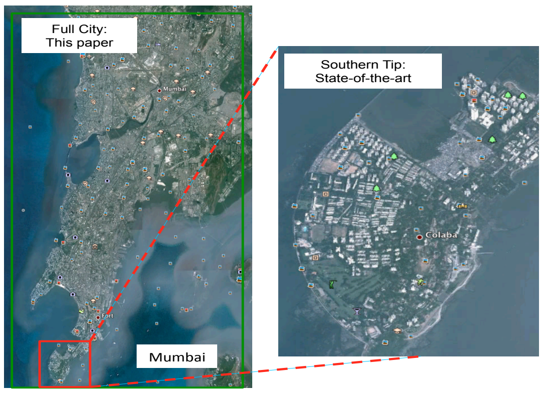


Figure 1: Comparison between SNARES and state-of-the-art: SNARES can now scale to solve problems the size of full cities where previous work could only scale to the southern tip of Mumbai.

Previous work has presented algorithms to compute approximate as well as optimal defender strategies for such domains [6, 12, 14]. RUGGED [6], the state-of-the-art optimal solver for this problem, is a double-oracle algorithm [8] that does not explicitly store the entire game matrix in memory. RUGGED computes the equilibrium strategy for a subset of the original game and then iteratively computes best responses of both the defender and the attacker to finally converge to a global equilibrium. However, two issues restrict RUGGED from scaling to real-world sized domains: (i) the best response modules are both NP-hard [6], and (ii) the large number of iterations required for the entire process.

The focus of this work is to present SNARES (Securing Networks by Applying a Randomized Emplacement Strategy), a new algorithm for computing the optimal solution for such domains. SNARES has two novel features. First, SNARES uses greedy heuristics to compute "better" (rather than "best") responses for *both players*. We show that the best-response problem for the defender has a sub-modularity property, and exploiting this provide a bound for the solution quality of our better response. Second, SNARES uses a novel "mincut-fanout" technique to warm-start the computation by initializing the game matrix with pure strategies for both the players. We show that naïve ways of warm-starting the computation can actually *increase* the runtime, and that our novel technique is effective. We extensively analyze these individual components of SNARES in this paper. Finally, we demonstrate SNARES's significant advance over the state-of-the-art: whereas state-of-the-art was restricted to the southern tip of Mumbai, with SNARES, optimal strategies for the entire road network of the city of Mumbai (refer Figure 1) can be obtained in a reasonable amount of time.

2. RELATED WORK

Many models have been proposed for network security domains. Typically called *pursuit-evasion* games, they have been specialized into many sub-classes based on mobility capabilities for both the players. For example, *hider-seeker* games [5] and *infiltration* games [1] allow both the attacker and defender to move around in the graph, whereas *search* games [3] have a fixed attacker and a mobile defender. Our work is similar to the sub-class of *interdiction* games [14] where the attacker is mobile and the defense measures are static. However, previous work either does not consider the presence of multiple targets with differing payoff values, or is not scalable to size of real-world networks.

More specifically, the following algorithms consider our network security model where the attacker chooses a path from a *source* to a *target* while the defender chooses to inspect edges. Washburn and Wood [14] showed that in the presence of exactly one target (or many targets with exactly the same payoff), a uniform distribution on the *mincut* between the sources and the target is the optimal strategy. However, this does not generalize to situations with multiple targets with differing payoff values. RANGER [12] is an algorithm that computes an approximate solution. It exploits the idea of *marginal* coverage and it was shown that the RANGER solution is optimal when the graph has certain structural properties. However, RANGER strategies can be arbitrarily bad in general, and even exhibited high error for real-world graphs [6]. Also, Jain et. al [6] presented RUGGED, which will be discussed in the next section.

Recent work has also generalized the network security problem to a non-zero sum game [10]; however, some of the key differences are in the modeling of the problem itself. For example, Okamoto et. al [10] consider the game between a *sender* sending data packets and an *attacker*, such that the expected utility to the sender is inversely proportional to the *harm* done by the attacker. The model used by this work allows the payoff function to be additive over the set of edges in the sender's path; however, such a model is not applicable in our domain where the payoff depends on whether or not the attacker is intercepted. An alternate line of research on network security is focused on developing algorithms that perform well against human subjects, by addressing human bounded rationality and human decision making [15]. However, these algorithms are not yet scalable to compute strategies for domains as large as real-world networks handled by SNARES.

3. PROBLEM DESCRIPTION

We borrow the description of the network security domain first introduced by Tsai et. al [12]. This domain is modeled using a graph $G = (\mathbf{N}, \mathbf{E})$. A pure strategy X_i for the defender is an allocation of the k defender resources to any k of the $|E|$ edges, i.e., $X_i = \{e_{i_1}, e_{i_2}, \ldots, e_{i_k}\}$. A pure strategy A_j for the attacker is a path starting at any one of the *source* nodes $s \in S$,[1] $S \subset N$ to any one of the targets $t \in T, T \subset N$. That is, $A_j = \{e_{j_1}, e_{j_2}, \ldots, e_{j_m}\}$ such that the source $s(j)$ of path A_j is also the source of edge e_{j_1} and the target $t(j)$ of path A_j is also the target of the last edge e_{j_m}. This leads to both the players having exponentially many pure strategies in this game, e.g., in a fully connected graph with 10 nodes, 1 target and 5 resources, there are $\binom{45}{5} \approx 1.2$ million pure strategies for the defender and approximately 4 million pure strategies for the attacker.

A payoff of $\mathcal{T}(t)$ is associated with each target $t \in T$, such that the defender gets $-\mathcal{T}(t)$ if the attacker successfully attacks t and 0 otherwise. Conversely, the attacker gains $\mathcal{T}(t)$ for a successful attack on target t and 0 on failure. Furthermore, success and failure of an attack is defined by computing the intersection between the pure strategy allocation of the defender and the pure strategy path of the attacker. For example, if edge e_3 was used in the attacker's path A_4 and was also covered in defender's allocation X_1, the attacker path A_4 is then a failure against the defender allocation X_1. Formally, the attacker fails along the path A_j against the defender allocation X_i if and only if $X_i \cap A_j \neq \emptyset$. The objective is to find a minimax strategy $\mathbf{x}$ for the defender (since this is a zero-sum game, minimax strategy is also a Nash equilibrium and a Strong Stackelberg equilibrium [16]).We describe the complete notation in Table 1.

Symbol	Meaning
$G(\mathbf{N}, \mathbf{E})$	Graph representing the network security problem.
$\mathcal{T}(t)$	Payoff for target t
k	Defender resources
$\mathbf{X}$	Set of defender allocations, $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$
$\mathbf{A}$	Set of attacker paths, $\mathbf{A} = \{A_1, A_2, \ldots, A_m\}$
$\mathbf{x}$	Defender's mixed strategy over $\mathbf{X}$
$\mathbf{a}$	Adversary's mixed strategy over $\mathbf{A}$
$U_d(\mathbf{x}, A_j)$	Defender's expected utility of playing $\mathbf{x}$ against A_j
$U_a(X_i, \mathbf{a})$	Attacker's expected utility of playing $\mathbf{a}$ against X_i

Table 1: The notation used in this paper is described here.

4. RUGGED

We now describe the RUGGED algorithm briefly since RUGGED is the principal competitor to SNARES. RUGGED is initialized with an arbitrary pure strategy $\mathbf{X} = \{X_1\}$ and $\mathbf{A} = \{A_1\}$ for both players. This set of pure strategies, $\langle \mathbf{X}, \mathbf{A} \rangle$ represents the current game on which RUGGED performs the computation. RUGGED uses Minimax and best response modules iteratively as follows: it first computes the minimax strategy $\mathbf{x}$ and $\mathbf{a}$ to the current game $\langle \mathbf{X}, \mathbf{A} \rangle$ (Line 4). This is followed by the best response module of the defender which computes X^*, or the defender's best response to $\mathbf{a}$ in Line 5. X^* is added to $\mathbf{X}$ if $X^* \notin \mathbf{X}$ (Line 6). This is followed

[1]Without loss of generality, we assume that $|S| = 1$, since in the presence of more than one source, a virtual source is added and connected to the existing sources.

by the best-response module of the attacker, which computes the best response A^* to $\mathbf{x}$ (Line 7). Again, analogous to the defender, A^* is added to $\mathbf{A}$ if $A^* \notin \mathbf{A}$. The algorithm converges if both X^* and A^* are already present in $\mathbf{X}$ and $\mathbf{A}$ respectively. Algorithm 1 provides a sketch for RUGGED.

Algorithm 1 RUGGED

```
1:  Initialize $\mathbf{X}$ using an arbitrary candidate defender allocation.
2:  Initialize $\mathbf{A}$ using an arbitrary candidate attacker path.
3:  repeat
4:      $(\mathbf{x}, \mathbf{a}) \leftarrow$ CoreLP$(\mathbf{X}, \mathbf{A})$.
5:      $X^* \leftarrow$ DO$(\mathbf{a})$.
6:      $\mathbf{X} \leftarrow \mathbf{X} \cup \{X^*\}$.
7:      $A^* \leftarrow$ AO$(\mathbf{x})$.
8:      $\mathbf{A} \leftarrow \mathbf{A} \cup \{A^*\}$.
9:  until convergence
10: return $(\mathbf{x}, \mathbf{a})$
```

Minimax: The minimax formulation (labeled `CoreLP`) used by RUGGED is given in Equations (1) to (4). Here, U_d^* represents the optimal (minimax) utility for the defender and $\mathbf{x}$ represents the defender's mixed strategy.

$$\max_{U_d^*, \mathbf{x}} \quad U_d^* \tag{1}$$

$$\text{s.t.} \quad U_d^* \leq U_d(\mathbf{x}, A_j) \quad \forall j = 1, \dots, |\mathbf{A}| \tag{2}$$

$$\mathbf{1}^{\mathrm{T}}\mathbf{x} = 1 \tag{3}$$

$$\mathbf{x} \in [0,1]^{|X|} \tag{4}$$

The mixed strategy $\mathbf{a}$ of the attacker can be obtained by computing the optimal values for the dual variables corresponding to the Equation family (2). The utility function $U_d(\mathbf{x}, A_j)$ evaluating the defender's mixed strategy $\mathbf{x}$ against attacker's pure strategy A_j is defined as in Equation (5), where $\mathcal{T}(t(j))$ is the payoff associated with the target $t(j)$ that is attacked through the path A_j.

$$U_d(\mathbf{x}, A_j) = -\mathcal{T}(t(j)) \cdot (\sum_i (1 - z_{ij})x_i) \tag{5}$$

Here, z_{ij} computes the intersection between the defender allocation X_i and attacker path A_j, and is given as

$$z_{ij} = \begin{cases} 1 & \text{if } X_i \cap A_j \neq \varnothing \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Defender Oracle: It computes X^*, or a best response of the defender *given the attacker's mixed strategy* $\mathbf{a}$ *as input*. This is done using the following MIP formulation (Equations (7)–(11)). Here again, $t(j)$ refers to the target being attacked through the path A_j. Also, A_j is represented as $\langle A_{je} \rangle$, where $A_{je} = 1$ if path A_j goes through edge e and 0 otherwise. Finally, X^* is constructed from the solution of the MIP by selecting edges e for which $\lambda_e = 1$, i.e., $X^* = \{e | \lambda_e = 1\}$. Here, the MIP formulation will set $z_j = 1$ if $X^* \cap A_j \neq \emptyset$, and 0 otherwise.

$$\max_{z, \lambda} \quad -\textstyle\sum_j (1 - z_j) a_j \mathcal{T}(t(j)) \tag{7}$$

$$\text{s.t.} \quad z_j \leq \sum_e A_{je}\lambda_e \tag{8}$$

$$\textstyle\sum_e \lambda_e \leq k \tag{9}$$

$$\lambda_e \in \{0,1\} \tag{10}$$

$$z_j \in [0,1] \tag{11}$$

Attacker Oracle: Similarly, the attacker oracle computes A^*, or the best response of the attacker *given the defender's mixed strategy* $\mathbf{x}$ *as input*. This is done using the following MIP formulation (one for each target), and then picking the best. The below formulation is for target t^*.

$$\max_{z, \gamma} \quad \mathcal{T}(t^*) \textstyle\sum_i x_i (1 - z_i) \tag{12}$$

$$\text{s.t.} \quad \textstyle\sum_{e \in \text{out}(n)} \gamma_e = \sum_{e \in \text{in}(n)} \gamma_e \quad n \neq s, t_m \tag{13}$$

$$\textstyle\sum_{e \in \text{out}(s)} \gamma_e = 1 \tag{14}$$

$$\textstyle\sum_{e \in \text{in}(t^*)} \gamma_e = 1 \tag{15}$$

$$z_i \geq \gamma_e X_{ie} \quad \forall e \forall i \tag{16}$$

$$z_i \in [0,1] \tag{17}$$

$$\gamma_e \in \{0,1\} \tag{18}$$

Again, $A^* = \{e | \gamma_e = 1\}$. This formulation sets $\gamma_e = 1$ such that the attacker follows a valid path along $e | \gamma_e = 1$ from the source to t^*. Also, z_i is set to 1 iff the attacker path $A^* \cap X_i \neq \emptyset$.

5. SNARES

We now present SNARES with the following three novel features: (1) an efficient warm start technique, (2) a better response heuristic for the defender, and (3) a better response heuristic for the attacker. The flow chart for SNARES is presented in Figure 2. The boxes with double lines are the three features mentioned above, and we will describe them in detail in this section. The formal algorithm is given as Algorithm 2.

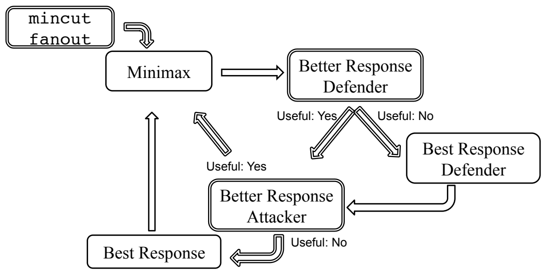


Figure 2: Flow Chart for the SNARES Algorithm

Algorithm 2 SNARES (G, k)

```
1:  Initialize $\mathbf{X}, \mathbf{A}$ using mincut-fanout
2:  repeat
3:      $(\mathbf{x}, \mathbf{a}) \leftarrow$ CoreLP$(\mathbf{X}, \mathbf{A})$.
4:      $X \leftarrow$ DBO$(\mathbf{a})$.
5:      if $U_d(X, \mathbf{a}) - U_d(\mathbf{x}, \mathbf{a}) \leq \epsilon$ then
6:          $X \leftarrow$ DO$(\mathbf{a})$.
7:      $\mathbf{X} \leftarrow \mathbf{X} \cup \{X\}$.
8:      $A \leftarrow$ ABO$(\mathbf{x})$.
9:      if $U_a(A, \mathbf{x}) - U_a(\mathbf{a}, \mathbf{x}) \leq \epsilon$ then
10:         $A \leftarrow$ AO$(\mathbf{x})$.
11:     $\mathbf{A} \leftarrow \mathbf{A} \cup \{A\}$.
12: until convergence
13: return $(\mathbf{x}, \mathbf{a})$
```

SNARES warm starts the computation with the pure strategies obtained using the `mincut-fanout` procedure, which will be explained next. `DBO` and `ABO` in Lines 4 and 8 refer to the defender better response and attacker better response oracles respectively, while `DO` and `AO` in Lines 6 and 10 are the best response oracles for both players respectively. In Line 4, `DBO` returns a heuristic response X of the defender (we call this heuristic response as a "better" response as opposed to the "best" response; here X may not be strictly better than existing strategies). Line 5 checks whether this utility $U_d(X, \mathbf{a})$ is higher than the utility $U_d(\mathbf{x}, \mathbf{a})$ obtained from minimax by at least ϵ, i.e., whether X is at least ϵ-better than all strategies present in $\mathbf{X}$ against $\mathbf{a}$. If $U_d(X, \mathbf{a})$ is not at least ϵ higher than $U_d(\mathbf{x}, \mathbf{a})$, then the best response DO is invoked (Line 6). Line 7 guarantees that each iteration adds an improving pure strategy to $\mathbf{X}$, should one exist. Similarly computation is performed for the attacker in Lines 8–11. Line 12 states that the computation proceeds until `convergence`, which is obtained when the utility obtained from the best response of each player is not better than the corresponding player's utility from the minimax strategy. In other words, `convergence` is obtained when $U_d(X, \mathbf{a}) - U_d(\mathbf{x}, \mathbf{a}) \leq \tau$ and $U_a(A, \mathbf{x}) - U_a(\mathbf{a}, \mathbf{x}) \leq \tau$, where τ defines the tolerance.[2] Finally, SNARES is guaranteed to converge on the global optimal solution since convergence can be obtained only when the best responses for both the players are unable to generate an improving strategy.

5.1 Warm-starting using `mincut-fanout`

The objective of warm-starting is to generate pure strategies for both the defender and the attacker before the computation of the minimax strategy is started. Given the setup of the game, the best response of the attacker will choose to attack the highest-valued target $\hat{t}$ with probability 1.0, if there exists a $s - \hat{t}$ path in $G(N, E)$ that does not intersect with any of the defender's allocations. Consequently, strategies considering the most-valued target get generated by the iterative procedure of the double-oracle based algorithm in the first few iterations. The objective of warm-starts here is to generate such strategies for both players and add them before the start of the double-oracle iterative procedure. This will reduce the number of iterations that are required by the algorithm.

Thus, we construct a game with just one target: $\hat{t}$. Solution for this game can be computed in polynomial-time: it is to uniformly distribute the defender's resources on the $s - \hat{t}$ min-cut [14]. Thus, `mincut-fanout` will sample pure strategies for the defender, or defender allocations, from the $s - \hat{t}$ min-cut, such that each allocation covers k edges. We then compute pure strategies for the attacker, or attacker paths, which are best responses to each individual defender allocation. This is done using Dijkstra's shortest path computation, such that each edge e in the defender allocation X has weight inf, while the other edges have a weight of 1. This also prefers short attacker paths over long ones, the intuition being that shorter paths should be preferred by the attacker since longer paths are more likely to be intercepted.

We also generalized the above idea to consider all the targets and not just the highest valued target $\hat{t}$. However, as we show in Section 6.1.1, our choice of using only the highest valued target in `mincut-fanout` performs better against considering all the targets. We also show that it performs better against other potential strategies for warm starting the computation.

5.2 Using "Better" Responses

SNARES presents heuristics to compute "better responses" for both players. Each better response module aims to compute a strategy for the corresponding player that is better than any strategy already in the mix against the other player's current equilibrium strategy. If successful, this guarantees that `CoreLP` will compute a different equilibrium when this strategy is added. If the better response heuristic fails to generate such a strategy, then the best-response module is called. The objective is to reduce the number of invocations of the best response modules, both of which solve an NP-hard problem [6] and consume significant runtime when the problem size gets bigger. On the other hand, the better response solutions used by SNARES can be computed in polynomial time. Furthermore, even with the use of better responses, SNARES still converges on the global equilibrium since the best responses modules are called if the better response modules do not generate an improving strategy. We now present the better response algorithms for both players.

5.2.1 *Better Response for the Defender*

In this section, we first present a greedy approach to generate the better response X_g of the defender. This approach greedily maximizes a "normalized" defender utility function $f(X, \mathbf{a})$. We next show that this utility function is a non-negative sub-modular function, and then establish a bound on the solution quality of our greedy better response solution. This bound suggests that our greedy approach generates good solutions.

The defender payoffs are always negative in our domain: the defender gets a payoff of $-\mathcal{T}(t)$ when the attacker successfully attacks target t and 0 otherwise. To facilitate analysis, we define a *non-negative normalized utility* function f for the defender, where

$$U_d(X, \mathbf{a}) \quad = -\textstyle\sum_{A_j \in \mathbf{A} | X \cap A_j = \emptyset} a_j \mathcal{T}(t_j) \tag{19}$$

$$f_{\mathbf{a}}(X) \qquad = U_d(X, \mathbf{a}) - U_d(\emptyset, \mathbf{a}) \tag{20}$$

More specifically, f gives the added benefit of the defender allocation X over the defender not protecting any edges. Furthermore, using Equation 19,

$$f_{\mathbf{a}}(X) = -\sum_{A_j \in \mathbf{A} | X \cap A_j = \emptyset} a_j \mathcal{T}(t(j)) - U_d(\emptyset, \mathbf{a}) = \sum_{A_j \in \mathbf{A} | X \cap A_j \neq \emptyset} a_j \mathcal{T}(t_j) \tag{21}$$

The greedy better response algorithm is described as Algorithm 3, where in each iteration, the objective is to add an edge e to the greedy defender allocation X_g that maximizes $f_{\mathbf{a}}(X \cup \{e\})$. Here, X_g (Line 1) is the computed better response of the defender. $\mathbf{a}$ is the mixed strategy of the attacker over the set of paths $\mathbf{A}$ that is input to the better response oracle. $\mathbf{A}_r$ in Line 2 is used to keep track of attacker paths that not already covered by the defender's allocation. w_e (initialized in Line 4) represents the weight of each edge. These weights are updated in Lines 5–7, such that w_e represents the total marginal usage of edge e in the attacker's mixed strategy $\mathbf{a}$, weighted by the payoff of the target attacked by the attacker when traversing through e. The defender, following the greedy approach, chooses an edge e^* with the highest weight in Line 8. Finally, all the paths intersecting with edge e^* are removed from the set of paths considered in subsequent iterations, i.e., from $\mathbf{A}_r$ (Lines 10–12). Lines 13–15 are invoked only if the allocation X_g already intersects with all the attacker paths $A_j \in \mathbf{A}$, and ensures that the defender chooses k edges in its allocation.

THEOREM 1. *The normalized defender utility, $f_{\mathbf{a}}(X)$, is submodular in X.*

PROOF. We show here that the gain in defender utility when adding an edge e to an existing defender allocation X exhibits diminishing returns. Let X_1 and X_2 be two defender allocations such that $X_1 \subseteq X_2 \subseteq E$. Furthermore, let $\mathbf{A}_1 = \{A_j | A_j \cap X_1 = \emptyset\}$, i.e., the set of attacker paths that are **not** covered by X_1. Similarly, let $\mathbf{A}_2 = \{A_j | A_j \cap X_2 = \emptyset\}$. Furthermore, $X_1 \subseteq X_2$, therefore, $\mathbf{A}_2 \subseteq \mathbf{A}_1$ since every path that is covered by X_1 is covered by X_2.

Moreover, using Equation 21,

$$f_{\mathbf{a}}(X_i) = -\sum_{A_j \in \mathbf{A}_i} a_j \mathcal{T}(t(j)) - U_d(\emptyset, \mathbf{a}), \quad i \in \{1, 2\} \tag{22}$$

[2]In all our experiments, we fix both ϵ and τ to 0.001.

Algorithm 3 Defender Better Response: $DBO(\mathbf{A}, \mathbf{a})$

```
1: Initialize $X_g \leftarrow \emptyset$ (defender allocation)
2: Initialize $\mathbf{A}_r \leftarrow \mathbf{A}$
3: while $|X_g| < k$ and $\mathbf{A}_r \neq \emptyset$ do
4:     $w_e = 0 \quad \forall e \in E$
5:     for all $A_j \in \mathbf{A}_r$ do
6:         for all $e \in A_j$ do
7:             $w_e \leftarrow w_e + a_j \mathcal{T}(t(j))$
8:     $e^* \leftarrow \arg\max_e w_e$
9:     $X_g \leftarrow X_g \cup \{e^*\}$
10:    for all $A_j \in \mathbf{A}_r$ do
11:        if $e^* \in A_j$ then
12:            $\mathbf{A}_r \leftarrow \mathbf{A}_r - \{A_j\}$
13: while $|X_g| < k$ do
14:    Choose $e$ arbitrarily from $E$
15:    $X_g \leftarrow X_g \cup e$
16: return $X_g$
```

Let us now consider the addition of an edge e. Let e intersect with attacker paths $A^e_{12} \cup A^e_2 \cup A^e_\emptyset$ where $A^e_{12} = \{A_j | e \in A_j, A_j \in \mathbf{A}, A_j \cap X_1 \neq \emptyset\}$ (and thus, paths in A^e_{12} also intersect with X_2), $A^e_2 = \{A_j | e \in A_j, A_j \in \mathbf{A}, A_j \cap X_1 = \emptyset, A_j \cap X_2 \neq \emptyset\}$, and $A^e_\emptyset = \{A_j | e \in A_j, A_j \in \mathbf{A}, A_j \cap X_2 = \emptyset\}$ (and thus, paths in $A^e_\emptyset$ also do not intersect with X_1). Here, $\mathbf{A}$ represents all the possible exponentially many attacker paths possible in the input network security game. Therefore,

$$f_{\mathbf{a}}(X_1 \cup \{e\}) - f_{\mathbf{a}}(X_1) \tag{23}$$
$$= \sum_{A_j \in A^e_2 \cup A^e_\emptyset} a_j \mathcal{T}(t(j)) \tag{24}$$
$$= \sum_{A_j \in A^e_2} a_j \mathcal{T}(t(j)) + \sum_{A_j \in A^e_\emptyset} a_j \mathcal{T}(t(j)) \tag{25}$$
$$= \sum_{A_j \in A^e_2} a_j \mathcal{T}(t(j)) + (f_{\mathbf{a}}(X_2 \cup \{e\}) - f_{\mathbf{a}}(X_2)) \tag{26}$$
$$\geq\ f_{\mathbf{a}}(X_2 \cup \{e\}) - f_{\mathbf{a}}(X_2) \tag{27}$$

Hence, the normalized defender utility $f_{\mathbf{a}}(X)$ is a non-negative sub-modular function. □

Letting X^* to be the best response of the defender,

$$f_{\mathbf{a}}(X_g) \geq (1 - \frac{1}{e}) f_{\mathbf{a}}(X^*) \tag{28}$$

since we compute an incrementally maximizing greedy solution to a non-negative sub-modular function [9].

We now establish the relationship between a global minimax equilibrium solution $\langle \mathbf{x}^*, \mathbf{a}^* \rangle$ and the solution $\langle \mathbf{x}_c, \mathbf{a}_c \rangle$ obtained when using just this greedy response DBO to compute pure strategies for the defender, i.e., we never call DO but we do call AO by taking out Lines 5 and 6 from Algorithm 2 to arrive at $\langle \mathbf{x}_c, \mathbf{a}_c \rangle$.

THEOREM 2. *The defender utility* $U_d(\mathbf{x}_c, \mathbf{a}_c)$ *is lower bounded by* $(1 - \frac{1}{e})U_d(\mathbf{x}^*, \mathbf{a}^*) + \frac{1}{e}U_d(\emptyset, \mathbf{a}_c)$.

PROOF. Firstly, given that $\langle \mathbf{x}^*, \mathbf{a}^* \rangle$ is a minimax solution,

$$U_d(\mathbf{x}^*, \mathbf{a}^*) \geq \quad U_d(\mathbf{x}, \mathbf{a}^*) \qquad \forall \mathbf{x} \tag{29}$$
$$U_d(\mathbf{x}^*, \mathbf{a}) \geq \quad U_d(\mathbf{x}^*, \mathbf{a}^*) \qquad \forall \mathbf{a} \tag{30}$$

Furthermore, defining

$$f_{\mathbf{a}}(\mathbf{x}) = \sum_{X_i \in \mathbf{X}} x_i f_{\mathbf{a}}(X_i) \tag{31}$$

and using Equations (20), (29), (30) and (31), we have:

$$f_{\mathbf{a}^*}(\mathbf{x}^*) \geq f_{\mathbf{a}^*}(\mathbf{x}) \qquad \forall \mathbf{x} \tag{32}$$
$$f_{\mathbf{a}}(\mathbf{x}^*) \geq f_{\mathbf{a}^*}(\mathbf{x}^*) + (U_d(\emptyset, \mathbf{a}^*) - U_d(\emptyset, \mathbf{a})) \qquad \forall \mathbf{a} \tag{33}$$

Therefore, using Equations (28) and (33),

$$f_{\mathbf{a_c}}(\mathbf{x}_c) = f_{\mathbf{a}_c}(DBO(\mathbf{a}_c)) \tag{34}$$
$$\geq (1 - \frac{1}{e}) f_{\mathbf{a}_c}(DO(\mathbf{a}_c)) \tag{35}$$
$$\geq (1 - \frac{1}{e}) f_{\mathbf{a}_c}(\mathbf{x}^*) \qquad \text{(by definition of } DO) \tag{36}$$
$$\geq (1 - \frac{1}{e})[f_{\mathbf{a}^*}(\mathbf{x}^*) + (U_d(\emptyset, \mathbf{a}^*) - U_d(\emptyset, \mathbf{a}_c))] \tag{37}$$

Therefore, $U_d(\mathbf{x}_c, \mathbf{a}_c) \geq (1 - \frac{1}{e})U_d(\mathbf{x}^*, \mathbf{a}^*) + \frac{1}{e}U_d(\emptyset, \mathbf{a}_c)$. □

Thus, not only is the better response defender utility bounded in each iteration, but the solution quality for using just the better response is also bounded at convergence. Furthermore, $U_d(\mathbf{x}_c, \mathbf{a}_c) \geq U_d(\mathbf{x}_c, \mathbf{a})\ \forall \mathbf{a}$. Thus, $U_d(\mathbf{x}_c, \mathbf{a}_c)$ is the utility that $\mathbf{x}_c$ guarantees the defender. Also, naturally, $U_d(\emptyset, \mathbf{a}_c) \geq -\max_{t \in T} \mathcal{T}(t)$. So the greedy solution does at least as well as doing nothing $\frac{1}{e}$ of the time and playing optimally the rest of the time. This proof suggests that the better response oracle can produce good solutions efficiently, a hypothesis which we experimentally validate in Section 6.

5.2.2 *Better Response for the Attacker*

We now describe the better response heuristic for the attacker. We use a shortest path based approach to generate better responses for the attacker, which is given as Algorithm 4. This algorithm is designed to accurately determine the defender's coverage probability when estimating the attacker's utility of the better response, even if the attacker chooses two edges in his path which are covered in the same defender allocation. For example, if in the attacker's better response, the attacker traverses through the edges e_1 and e_2, and there exists a defender allocation $X_i | e_1, e_2 \in X_i$, then Algorithm 4 will not double count the probability x_i associated with allocation X_i when computing the attacker's reward. This is different from previous greedy approaches for computing the attacker's response, which defined the cost of the edge e as the defender's marginal coverage of e [12], and suffered from over-estimation of the defender's coverage. Algorithm 4 below assumes the presence of only one source s; if there are multiple sources, then a virtual source is added which then connects to all the existing sources.

Lines 1–15 of Algorithm 4 follow the Dijkstra's algorithm. Here, path distances are computed using `caught[`u`]`, which gives the probability of the attacker's $s - u$ path getting intercepted by the defender. To ensure the correct computation of `caught`$[u]$, the algorithm keeps track of $\overline{\mathbf{X}_u}$, or the set of allocations that the attacker has *not* encountered along the $s - u$ path. $\overline{\mathbf{X}_v}$ is updated once the attacker moves to node v using the edge $e : (u, v)$ (Line 14), by removing all the allocations from $\overline{\mathbf{X}_u}$ that contributed to the cost c_e of edge e. c_e (Line 10) gives the probability of the attacker getting intercepted by the defender on this particular edge, only considering allocations in $\overline{\mathbf{X}_u}$. Lines 16 to 18 then find highest expected utility target to attack for the attacker, and the greedy path A_g is then constructed using the stored predecessor information (Lines 13 and 19). As opposed to the defender's pure strategies, the edges for the attacker *cannot* be chosen independently, because they should define a valid path from s to a target t. Such a restriction prevents us from conducting a sub-modularity analysis similar to the defender case; and we focus on experimental validation of this approach. The results are presented in Section 6.

Algorithm 4 Attacker Better Response: $ABO(\mathbf{X}, \mathbf{x})$

```
1: for all $n \in N$ do
2:     caught[$n$] ← $\infty$
3: caught[$s$] ← 0
4: $\overline{\mathbf{X}_s} \leftarrow \mathbf{X}$
5: Add $s$ to Priority Queue PQ
6: while PQ is not empty do
7:     $u \leftarrow \arg\min_{n \in PQ}$ caught[n]
8:     Remove $u$ from PQ.
9:     for $e(u, v) \in$ out-edges($u$) do
10:        $c_e \leftarrow \sum_{X_i \in \overline{\mathbf{X}_u} | e \in X_i} x_i$
11:        if caught[$u$] + $c_e \leq$ caught[$v$] then
12:            caught[$v$] ← caught[$u$] + $c_e$
13:            prev[$v$] ← $u$
14:            $\overline{\mathbf{X}_v} \leftarrow \overline{\mathbf{X}_u} - \{X_i | e \in X_i\}$
15:            Add $v$ to PQ
16: for $t \in T$ do
17:     payoff[$t$] ← $(1 -$ caught[$t$]$) \cdot \mathcal{T}(t)$
18: $t^* \leftarrow \arg\max_t$ payoff[$t$]
19: $A_g \leftarrow$ path($s - t^*$) constructed using prev[$t^*$]
20: return $A_g$
```

6. EXPERIMENTAL RESULTS

We now experimentally evaluate the performance of SNARES, both on simulated graphs as well as on real urban road networks.

6.1 Analysis of Components of SNARES

In this section, we evaluate the performance boost provided by each component of the SNARES algorithm. Specifically, we compare the performance with and without the use of `mincut-fanout` strategies for warm starts as well as with and without the better responses. These experiments were conducted on a machine with 16 GB main memory and a 2.3 GHz processor.

6.1.1 *Warm Starts without Better Responses:*

We compare the performance of choosing the `mincut-fanout` warm-start methodology with 5 other methodologies, ranging from random selection to using previously published algorithms for this domain. We also establish the baseline using RUGGED.

All data points are averaged over 30 samples for random geometric graphs. We use random geometric graphs since they have been shown to mimic the connectivity properties of real road networks [2]. A random geometric graph is generated as follows: nodes or vertices are placed at random uniformly and independently on a 2-D region, and two vertices u, v are connected by an edge if and only if the distance between them is at most a threshold d, i.e., $||x - y||_2 \leq d$. In all our experiments, our 2-D region is normalized to a unit square, and the value of d varies from 0 to 1.

These results are shown in Figure 3(a). The y-axis shows the runtime in seconds and the x-axis shows the different methodologies for warm starts. These results are averaged over 30 random geometric graphs with 50 nodes, 5 targets, 3 defender resources and $d = 0.2$. The target payoffs were randomly generated between 0 and 100. The first bar of the graph represents the runtime of RUGGED. The second, third and fourth bars are results when warm starts are used for the defender. They represent a random choice of k edges, sampling of defender pure strategies from the union of min-cuts between the source s and each target $t \in T$, and pure strategies obtained by sampling the solution obtained using RANGER respectively. The fifth and the sixth bars are results when warm-starts are used for the attacker. They use the RANGER algorithm and shortest paths from the source s to each target $t \in T$ respectively. The seventh bar in the graph represents the results of using `mincut-fanout` as in SNARES. Here, RUGGED took 329.69 seconds, random selection of edges for the defender increased the runtime to 435.24 seconds whereas `mincut-fanout` reduced it to 76.67 seconds. These results show that while `mincut-fanout` is effective, other approaches are not as effective and may even perform worse than RUGGED.

6.1.2 *Better Responses without Warm Starts*

We now present the results of using better responses in SNARES. We evaluate the use of better responses for each player independently as well as in conjunction. However, no warm starts were used in these experiments, that is, the full Algorithm 2 was used but with Line 1 disabled. These experiments are also done on the same graphs as before: 30 samples of random geometric 50 node graphs with 5 targets, 3 defender resources and $d = 0.2$. Again, RUGGED forms the baseline. These results are shown in Figure 3(b). The y-axis shows the runtime in seconds whereas the x-axis shows the different configurations for the experiment. For example, RUGGED took 329.69 seconds whereas using better responses for just the defender reduced the runtime to 33.58 seconds. Moreover, SNARES required just 4.46 seconds, an improvement of 98%. These results also show that using better responses for any one player provides a boost in performance, and using it for both players simultaneously makes SNARES even more effective.

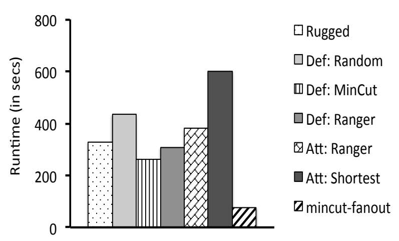
Figure 3(c) shows the percentage of times calls to the best response module have to be made on the y-axis (on log-scale) and varies the configuration on the x-axis. The results are shown for all the four configurations from the previous experiment. Each result shows the percentage of iterations better response did *not* generate an improving strategy for both the players (i.e, in Algorithm 2, check in Line 5 failed and Line 6 was called for the defender, and check in Line 9 failed and Line 10 was called for the attacker). The lower the bar, the lower percentage of times the best response module has to be called. RUGGED has no better responses, and is hence plotted at 100%. For example, the best response of the attacker and defender were computed in only 15.81% and 1.69% of the iterations in SNARES.

6.2 Scalability in Simulation

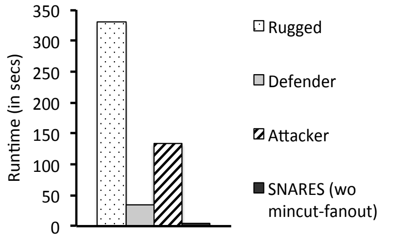
This section evaluates the performance of SNARES as the input problem size is varied. These results are also conducted on random geometric graphs and are averaged over 30 samples. We do not plot error bars in our graphs because the y-axis is on a log-scale; however, SNARES was statistically significantly (with $p < 0.05$) faster than RUGGED for all experiments with more than 100 nodes, or $d \geq 0.2$, or $k \geq 4$, or $|T| \geq 4$.

6.2.1 *Vary Number of Nodes*

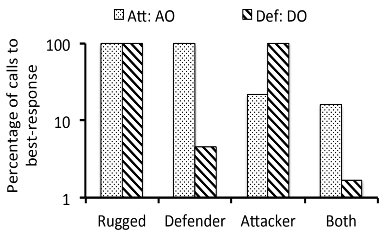
Figure 4(a) presents the results when varying the number of nodes in the input problem. The x-axis shows the number of nodes in the graph, whereas the y-axis shows the runtime in seconds (on a log-scale). The four bars in the graph compare the performance of RUGGED with (i) SNARES without better responses, (ii) SNARES without `mincut-fanout`, and (iii) SNARES. These experiments were conducted on random graphs generated with density $d = 0.1$, 3 targets, 3 source nodes for the attacker, and 3 defender resources. These results show that all configurations are better than the baseline of RUGGED, and that SNARES is the most efficient. RUGGED ran out of memory in 26 out of 30 samples for 200 node graphs, and was killed in another 2 samples since it did not finish execution in 3 hours. For example, for 150 nodes, while RUGGED took 6021.08



(a) Effect of warm starts

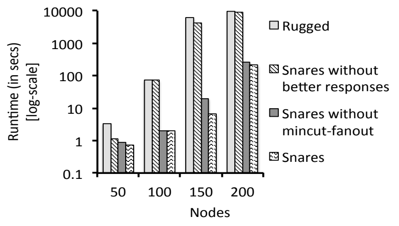


(b) Effect of using better responses

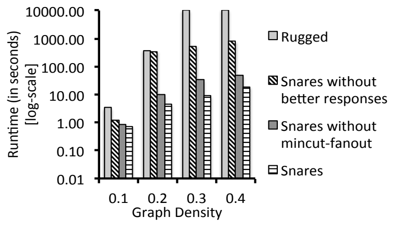


(c) Analyzing better responses

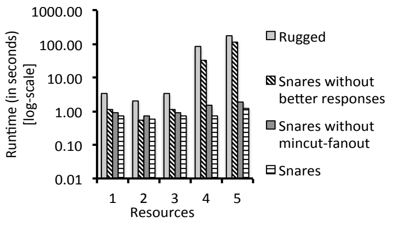
Figure 3: The contributions of individual components of SNARES. RUGGED is used as a baseline.



(a) Varying number of nodes



(b) Varying density d



(c) Varying number of defender resources

Figure 4: The runtime required by SNARES as the input problem size is varied.

seconds, and SNARES without better responses used $4,152.80$ seconds. Additionally, SNARES without `mincut-fanout` reduced the runtime to 19.58 seconds and SNARES required only 6.53 seconds. Thus, the experiment shows that the combination provides the most significant boost in performance, SNARES taking only 1.08% of the time required by RUGGED.

6.2.2 Vary Graph Density

Figure 4(b) presents the results when varying the distance d used when generating the random geometric graphs. The x-axis shows the value of d, whereas the y-axis gives the runtime. These experiments were conducted on random graphs generated with density 50 nodes, 3 targets, 3 source nodes for the attacker, and 3 defender resources. These results show that as the density of the graph is increased, the runtime required by all the algorithms increases. Experiments that did not terminate in $10,800$ seconds (3 hours) were terminated (as in the case of most experiments with graph density higher than 0.3 when run with RUGGED). These results also show that SNARES performs much better than RUGGED. For example, for $d = 0.3$, most experiments with RUGGED did not finish in 3 hours, whereas SNARES without better responses took 537.19 seconds. Furthermore, SNARES without `mincut-fanout` required 32.74 seconds, whereas SNARES required only 9.04 seconds.

6.2.3 Vary Number of Resources

We now present the results when varying the number of defender resources in Figure 4(c). These results are shown for random graphs with 50 nodes, 3 targets, 3 sources and graph density $d = 0.1$. The x-axis in this graph shows the number of defender resources, whereas the y-axis shows the runtime in seconds on log scale. The results show that the performance trends remain the same: SNARES scales much better than RUGGED when the problem size is increased and the use of better responses provides a more significant performance boost. For example, for 4 resources, RUGGED required 83.90 seconds whereas SNARES only required 0.72 seconds on average.

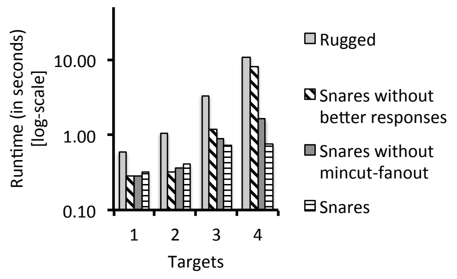


Figure 5: Varying Number of targets.

6.2.4 Vary Number of Targets

Figure 5 presents the results when varying the number of targets for random graphs 50 nodes, 3 sources, 3 resources and $d = 0.1$. The x-axis shows the number of targets, whereas the y-axis shows the runtime in seconds on log-scale. We observe similar performance trends: SNARES scales better than RUGGED in the number of targets as well. For example, for 4 targets, where RUGGED took 10.97 seconds, SNARES only required 0.77 seconds.

6.3 Real Data

We also tested the performance of SNARES on real urban road network data. We downloaded the OSM information for the road network of Mumbai [4]. We extracted this information for the entire city of Mumbai, that is, the road network existing between latitudes 18.840 and 72.750 and longitudes 19.360 and 73.160.

We first present results in Table 2 comparing SNARES with RUGGED on the southern part of the Mumbai road network. RUGGED was only able to solve for a subset of the Mumbai road network, and thus, we can only use a subset of the Mumbai map for this experiment. These experiments were done with 3 targets and 3 sources, whose locations on the map were motivated by the Mumbai attacks of November 2008. These results show that SNARES has a signif-

Map		Resources			
Size	Algorithm	1	2	3	4
45	RUGGED	0.91	6.43	22.58	33.42
	SNARES	1.08	2.62	7.53	7.71
129	RUGGED	6.63	32.55	486.48	3,140.23
	SNARES	2.23	2.99	10.99	21.06
252	RUGGED	17.19	626.25	2,014.14	34,344.70
	SNARES	3.83	4.85	16.19	30.77

Table 2: Runtime of RUGGED [6] and SNARES on real Mumbai map. Results of SNARES are averaged over 30 runs.

icant improvement over RUGGED; for example, for a subset of the graph with 252 nodes, RUGGED took 34,344.70 seconds to place 4 resources whereas SNARES only 30.77 seconds!

While RUGGED could only compute solutions for the southern tip of the road network of Mumbai, SNARES was able to solve the entire road network. These results are shown in Table 3. We ran SNARES varying the number of targets and number of defender resources for 3 sources. The easy-hard-easy pattern in the runtime with the increase in resources is expected based on the runtime properties of security games [7]. For example, it took SNARES only 101.09 seconds to find the optimal solution for placing 10 checkpoints when considering 8 targets. Thus, SNARES is now scalable enough to be applied and used in the real world.

Resources	Number of Targets	
	4	8
1	18.23	27.12
5	1,209.37	7,289.03
10	27.26	129.51
15	12.64	101.09

Runtime Required by SNARES: All results averaged over 30 samples.

Table 3: The image is map of Mumbai road network comprising 9,503 nodes and 20,416 edges. The sources (blue dots) and targets (red dots) are placed arbitrarily for these tests.

7. CONCLUSIONS

Scheduling of defender resources on a network is a significant and challenging research problem with real-world importance. Faster and more scalable security algorithms are required before such algorithms can be of assistance to end-users in the real world. This paper presents a significant advance towards addressing this challenge. For example, whereas previous state-of-the-art algorithms could solve for the road network of at most the southern tip of the city of Mumbai, we can now solve for the entire city providing orders-of-magnitude speedups. This advance is made possible by SNARES, which makes the following new contributions: First, we provide `mincut-fanout`, an effective procedure to warm start the computation. Second, we show that the defender optimization exhibits the sub-modularity property and we exploit it by developing a greedy better-response heuristic. We also present a better-response heuristic for the attacker. These methods combine providing SNARES orders of magnitude improvement over previous algorithms, as we show in extensive analysis of individual components of SNARES. Third, we also present results on much larger real-world urban road network data, and demonstrate that SNARES can now compute solutions for problems of the size encountered in the real-world. The scalability of SNARES now makes it feasible to use game-theoretic strategies in real-world networked domains.

8. ACKNOWLEDGEMENTS

This research has been funded by the Army Research Office through the MURI grant number W911NF-11-1-0332.

9. REFERENCES

[1] S. Alpern. Infiltration Games on Arbitrary Graphs. *Journal of Mathematical Analysis and Applications*, 163:286–288, 1992.

[2] D. Eppstein and M. T. Goodrich. Studying (Non-Planar) Road Networks Through an Algorithmic Lens. *CoRR*, abs/0808.3694, 2008.

[3] S. Gal. *Search Games*. Academic Press, New York, 1980.

[4] M. Haklay and P. Weber. Openstreetmap:user-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, 2008.

[5] E. Halvorson, V. Conitzer, and R. Parr. Multi-step Multi-sensor Hider-Seeker Games. In *IJCAI*, pages 159–166, 2009.

[6] M. Jain, D. Korzhyk, O. Vanek, V. Conitzer, M. Pechoucek, and M. Tambe. A Double Oracle Algorithm for Zero-Sum Security Games on Graphs. In *AAMAS*, 2011.

[7] M. Jain, K. Leyton-Brown, and M. Tambe. The Deployment-to-Saturation Ratio in Security Games. In *AAAI*, 2012.

[8] H. B. McMahan, G. J. Gordon, and A. Blum. Planning in the Presence of Cost Functions Controlled by an Adversary. In *ICML*, pages 536–543, 2003.

[9] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An Analysis of Approximations for Maximizing Submodular Set Functions–I. *Mathematical Programming*, 14(1):265–294, Dec 1978.

[10] S. Okamoto, N. Hazon, and K. Sycara. Solving non-zero sum multiagent network flow security games with attack costs. In *AAMAS*, pages 879–888, 2012.

[11] E. Shieh, B. An, R. Yang, M. Tambe, C. Baldwin, J. DiRenzo, B. Maule, and G. Meyer. Protect: A Deployed Game Theoretic System to Protect the Ports of the United States. In *AAMAS*, 2012.

[12] J. Tsai, Z. Yin, J. young Kwak, D. Kempe, C. Kiekintveld, and M. Tambe. Urban Security: Game-Theoretic Resource Allocation in Networked Physical Domains. In *AAAI*, pages 881–886, 2010.

[13] O. Vanek, Z. Yin, M. Jain, B. Bosansky, M. Tambe, and M. Pechoucek. Game-theoretic resource allocation for malicious packet detection in computer networks. In *AAMAS*, 2012.

[14] A. Washburn and K. Wood. Two-person Zero-sum Games for Network Interdiction. *Operations Research*, 43(2):243–251, 1995.

[15] R. Yang, F. Fang, A. X. Jiang, K. Rajagopal, M. Tambe, and R. Maheswaran. Designing Better Strategies against Human Adversaries in Network Security Games. In *AAMAS (Extended Abstract)*, 2012.

[16] Z. Yin, D. Korzhyk, C. Kiekintveld, V. Conitzer, and M. Tambe. Stackelberg vs. Nash in Security Games: Interchangeability, Equivalence, and Uniqueness. In *AAMAS*, pages 1139–1146, 2010.